USER'S GUIDE TO CLTC

A Software for MCMC Convergence Diagnostic
using the Central Limit Theorem for Markov chains

www.univ-orleans.fr/mapmo/membres/chauveau/pgm/cltc/cltc.html

Note: This program is available freely for **non-commercial** use only

Version 7.5 (2002)

Didier CHAUVEAU

Université d'Orléans, France

didier.chauveau@univ-orleans.fr

Contents

1 Introduction **3**

1.1 MCMC methods 4

1.2 MCMC convergence assessment 4

1.3 Methodology behind the CLTC software 5

1.4 Overview of the CLTC software 5

1.4.1 Method 1: Running CLTC using observations stored in a file (preferable) 7

1.4.2 Method 2: Plugging the MCMC code into CLTC 8

1.4.3 CLTC output and output files 8

2 Running CLTC **9**

2.1 The CLTC distribution 9

2.2 Preparing for Method 1: control after the simulation 10

2.2.1 Using the "single to parallel" converter 12

2.3 Preparing for Method 2: simulation and control in the same program 12

2.3.1 Including the MCMC algorithm 13

2.4 Compiling 14

2.5 Input files and CLTC tuning parameters 15

2.5.1 The discrete case 16

2.6 Launch CLTC 16

2.7 Program output 17

2.7.1 The C program output log 17

2.7.2 The `CLTC` output file 19

2.7.3 The Mathematica output 20

2.8 Interacting with the program 21

Chapter 1

Introduction

This document is a user's guide for a software called "CLTC" (an acronym for Control via the Central Limit Theorem), which implements the methodology described in Chauveau and Diebolt [4]. The objective of this software is

Monte Carlo Markov Chains (MCMC) convergence assessment.

Note that this software will *not* help you to design and implement an MCMC algorithm for your particular application. The convergence diagnostic is a calculation that is typically performed after or along with the simulations. CLTC provides you with a generic, "black-box" type software tool for the convergence diagnostic of your MCMC algorithm. It can be used to perform the convergence diagnostic simultaneously with the simulations, or as a second step after the simulations (provided that you ran a sufficiently large number of iterations from your MCMC sampler and that you stored these observations in a file).

This guide is intended to help potential users to run this software. Its goal is therefore *not* to explain what are MCMC algorithms, and what is the convergence assessment problem. References for the theoretical background are given at the end of the document. Good introduction to MCMC methods are [7] or [10], and only the very basics are recalled briefly below. For the same reason, the methodology behind the CLTC software is not entirely described here. Users are strongly encouraged to take a look at Chauveau and Diebolt [4], or alternatively to check the companion preprint [2] that is included in the distribution (as the postscript file `CLTCtechreport.ps`). This toolbox is available online and can be sent to interested users upon request to `didier.chauveau@univ-orleans.fr`. Other explanations and downloads can eventually be found on the web URL:

`www.univ-orleans.fr/mapmo/membres/chauveau/pgm/cltc/cltc.html`

This program is available freely, but is for **non-commercial** use only.

1.1 MCMC methods

In short, a MCMC method is a way to define a Markov chain $(\theta_t)_{t\geq 0}$ with some given probability distribution π as its stationary distribution. This *target distribution* usually comes as the posterior $\pi(\theta|\mathbf{x})$ for the parameter θ is some Bayesian setup given the data $\mathbf{x}$. Hence π is known only up to a multiplicative constant, and i.i.d. sampling from π or inference that requires computation of expectations like $\mathbb{E}_\pi(h) = \int h(\theta)\pi(\theta)\,d\theta$ are not directly available. We assume that h is real-valued to simplify the presentation given here.

The Markov chains produced by MCMC methods are ergodic under mild conditions, so that the Strong Law of Large Numbers (SLLN) holds. This allows the approximation of $\mathbb{E}_\pi(h)$ by the empirical average

$$\frac{1}{n}\sum_{t=1}^{n} h(\theta_t). \tag{1.1}$$

The Central Limit Theorem (CLT) for Markov chains holds if there exists a limiting variance

$$\sigma^2(h) = \lim_{n\to\infty} \frac{1}{n}\mathrm{var}\left[S_n(h)\right], \tag{1.2}$$

satisfying $0 < \sigma^2(h) < +\infty$, and such that

$$\frac{1}{\sqrt{n}} S_n(\bar{h}) \xrightarrow{d} \mathcal{N}\left(0, \sigma^2(h)\right), \tag{1.3}$$

where

$$S_n(h) = \sum_{t=1}^{n} h(\theta_t) \quad \text{and} \quad S_n(\bar{h}) = \sum_{t=1}^{n} \left[h(\theta_t) - \mathbb{E}_\pi(h)\right].$$

Conditions for the CLT to apply are given, e.g., in Meyn and Tweedie [9]. When the CLT holds, the precision of the approximation (1.1) can be computed, or confidence intervals can be proposed. Producing such confidence intervals for parameter estimates is one of the by-products of the `CLTC` software.

1.2 MCMC convergence assessment

The conditions under which the Markov chains produced by MCMC methods are ergodic are necessary, but however insufficient for the implementation, since they suffer from a significant problem in actual application: how to determine the moment when we can conclude their convergence, in other words, when one should stop the chain and use the observations in order to estimate the distribution characteristics considering that the sample is sufficiently representative of the stationary distribution. This is the so-called *convergence assessment* problem. More details about convergence assessment, together with a survey of the literature on the subject can be found in Brooks and Roberts [1].

An important criterion for convergence assessment methods is the required computer investment: diagnosis requiring problem-specific computer codes for

their implementation (e.g., requiring knowledge of the transition kernel of the Markov chain) are far less usable for the end user than diagnosis solely based upon the output of the sampler, since the latter can use available generic code. The `CLTC` software is completely generic, since it is based only on the realisations from parallel chains, and it works without knowledge on the sampler driving the chain. In addition, the normality diagnosis leads to automated stopping rules. This is the reason why it has been implemented in a software available online.

1.3 Methodology behind the CLTC software

The methodology described in [4] or [2] is grounded on the fact that approximate normality of appropriate functions of the chain is an indication of sufficient mixing. The diagnostic is based on the time needed to reach approximate normality for normalized sums of functions of the Markov chain like $S_n(h_r)/\sqrt{n}$, for a collection of real-valued functions h_r, $r = 1, 2, \ldots$ related to the posterior distribution or the parameters of interest. This convergence time is obtained by `CLTC` through statistical tests and variance monitoring performed over time and based on the realisations from **independent parallel (i.i.d.) chains**, i.e. m simulated i.i.d. copies of the chain:

$$\theta^{(\ell)}, \ell = 1, \ldots, m, \quad \text{where} \quad \theta^{(\ell)} = \theta_t^{(\ell)}, t = 1, \ldots, n \quad \text{is the } \ell\text{-th chain.}$$

The first control tool tests the normality hypothesis on the basis of the i.i.d. samples of these normalized averages computed over the i.i.d. chains. A second connected tool is based on graphical monitoring of the stabilization of the associated limiting variance. A first set of functions h's consist in indicator functions $h_r = \mathbb{I}_{A_r}$ of subsets A_r of the support of the target π. This gives convergence diagnostic, approximate normality and confidence intervals for the histogram (the picture) of π. A second set consists in moment functions like (in dimension one) $h(\theta) = \theta^p$, $p = 1, 2, \ldots$. This gives convergence diagnostic and confidence intervals for the parameter moments, e.g., for means and variances.

As stated previously, `CLTC` can run the parallel simulation for you and compute the convergence time during the simulation (for that you just have to define your MCMC sampler within the source code and compile the whole thing). Alternatively, you can run your own program for your MCMC sampler, store the realisations from i.i.d. chains in a plain text file, and run a generic version of `CLTC` using this file as an input file. These two methods are decribed in the next section.

1.4 Overview of the CLTC software

Theoretically, the methodology used by `CLTC` can be applied to any Markov chain in any general state space. However, the convergence diagnostic of the histogram of π (i.e. over the indicator functions of subsets of the state space)

is hard to implement as dimension increases. For simplicity and ease of implementation, we have thus restricted the generic `CLTC` implementation to control **one-dimensional functions of the original chain**. Functions of Markov chains are not Markov chains, but should satisfy the CLT if the original chain does, and hence are still testable for sufficient mixing using `CLTC`. The method then consists in

diagnosing convergence for each marginal of the original chain,

and eventually for any suitable one-dimensional function (e.g. any recoding of interest) of the original chain

$$\big\{\psi(\theta_t)\big\}, \quad t \geq 0, \quad \text{for } \psi(\cdot) \in \mathbb{R}.$$

The fact that all the marginals satisfy the normality convergence criterion computed by `CLTC` does not mean that the multivariate original chain satisfies the criterion after the same number of iterations. This is a trade-off between simplicity/ease of implementation and precision or conservative rule. Advantages and drawbacks of this approach are discussed in [4].

The main features of the program are:

- convergence diagnostic for both discrete and continuous Markov chains;
- automated control diagnostics, based on simple tuning parameters;
- "black box" tool, completely independent from the MCMC algorithm to control;
- does not require technical/analytical knowledge of the kernel driving the chain;
- standard `ANSI C` generic code for the essential part of the software (from parallel simulations to diagnostic);
- additional (optional) features available in a *Mathematica* NoteBook document, delivering graphics for visual control and more diagnostics based on asymptotic variance stabilization.

The drawback of this methodology is the computing time: `CLTC` needs simulation of m parallel (i.i.d.) chains, even if this is not required by the application. Note that for technical reasons

m **must be 30 or 50 in this implementation.**

Since the control method is generic, the "core" source code of the program **should not be edited or modified by the user**. However, the `CLTC` black box needs to be "linked" in some way to the output from the MCMC algorithm which needs a convergence diagnostic. This may be done in two ways:

1.4.1 Method 1: Running CLTC using observations stored in a file (preferable)

The first way of using `CLTC` is by writing simulation output from your MCMC algorithm in a plain text file (in a simple format that will be specified in § 2.2), and by "telling" `CLTC` where the file is at runtime. Since the method is using parallel chains, `CLTC` needs an "array" of simulated values, i.e. the m i.i.d. chains running for n iterations. Fortunately, you don't really need to write a parallel version of your MCMC sampler. It is often simpler to run your "single chain" program m times and to store the n iterations of each run in a corresponding plain text file. It is then easy to write a little piece of code to bring together these m single chain files into one **"parallel chains file"** suitable for input to `CLTC`. A generic program for doing this (file `single2parallel.c`) is already included in the distribution and is discussed in § 2.2.1. The only important thing is that **the m initial parameter values $\theta_0^{(\ell)}$ must be drawn randomly from a dispersed distribution** over the support of π. That is, **the m chains should not start from the same initial position**.

Advantages of using Method 1 are that:

1. it is the easiest way when the MCMC algorithm is not originally implemented in `ANSI C` or `C++`;
2. the `CLTC` source files need not to be edited at all: a ready-to-use set of source files (folder `CLTC_generic_Method1`) for one-dimensional realisations (function or marginal of the original Markov chain) is already included in the distribution; an already compiled application is included in the Macintosh version;
3. the computing time necessary to run `CLTC_1D_generic` itself when the simulations are already done and available in file is very small (less than 1 mn for $m = 50$ chains of $n = 10,000$ iterations each, on an average PowerMac G3). This allows to quickly check several settings for the tuning parameters on the same simulations output (see § 2.5 and [2] or [4]);
4. if the MCMC is complex (time consuming simulation, high dimension) you can just run the m simulations once, and write the output of all the marginals (and/or any suitable functions $\psi(\theta)$) in separate files during the simulation; then you have to run `CLTC_1D_generic` over each "parallel chains file" (one for each marginal), but these steps are not CPU demanding (see point 3).
5. if, when running `CLTC_1D_generic` over the file containing the n iterations for the m chains of your sampler, convergence is not reached, you can run more iterations for the m chains with $\theta_n^{(\ell)}$ as the starting value for chain ℓ, and append these more iterations to the original file. Then re-run `CLTC_1D_generic` and so on, up to convergence.

1.4.2 Method 2: Plugging the MCMC code into CLTC

The original idea behind the implementation of `CLTC` was that, since the user needs in any way to write the piece of code implementing the MCMC method for its particular application, it is convenient to plug this piece of code into another generic piece of code that will do the necessary simulation and the convergence diagnostic simultaneously, and will call the MCMC sampler when appropriate. This gives another way of using `CLTC`. Advantages of this approach are that:

1. the user needs only to implement a `C` procedure, performing one step of the MCMC sampler[1], to adjust some external variables, and to compile this code together with the `C` package given in the distribution (the code is already prepared for this).
2. the user needs not to implement the parallel simulation; these are done automatically at runtime by `CLTC`.

A drawback of this approach is that `CLTC` needs to be run for each marginal, since it controls one marginal at a time. If the MCMC is complex (i.e. CPU time-consuming), or in high dimension, it is generally preferable to use the first method.

1.4.3 CLTC output and output files

The `CLTC` software consists in two parts: a program in `ANSI C` for the heavy simulation and convergence diagnostic task, and a *Mathematica* NoteBook[2] document for some user-friendly and interactive output, graphics monitoring convergence and improved convergence diagnostic that needs some matrices inversion (see § 2.4 and [4]). The *Mathematica* processing in not required; you may just use the `C` program if you just don't need graphics and improved convergence tool, or if you cannot run *Mathematica* on your system.

The `C` program gives an output in the window where it has been launched (`UNIX`) or in the window it opens when launched (`MacOS`). This ouput gives the result of the convergence diagnostic and some useful information about the controlled region, its mass, the controlled mass, etc... (see § 2.7, and [2] or [4] for a complete description). The `C` program also writes an output file which is actually a *Mathematica* document that can be processed with the NoteBook `CLTC_public_7.5.nb` included in the distribution to deliver graphics, etc... (see § 2.7 and information on the NoteBook itself).

[1] typically called `MCMC(current_theta,next_theta)`, where the input argument is the current value, and the output the next value of the multivariate parameter

[2] *Mathematica* 3 or higher is preferable; check `http://www.wolfram.com`

Chapter 2

Running CLTC

2.1 The CLTC distribution

`CLTC` is available for both `UNIX` and Macintosh platforms. Actually, it can be used on Wintel platforms also since it only consists in `C` sources and *Mathematica* NoteBooks that are plain text files. You only need a `C` compiler and a running *Mathematica* system. The distribution consists in several `C` files, some plain text sample input and output files, two *Mathematica* Notebook files, and the companion preprint and this document as PostScript files. The Macintosh version also includes two already compiled executables: `CLTC_1D_generic` for the generic version (i.e. using Method 1), and another one implementing a sample version of `CLTC` using Method 2, for a simple Gibbs sampler that has been used in [3]. Executable files are not provided for the `UNIX` version since the compiled code depends on the platform (SUN, HP workstations, PC running `LINUX`, etc...). Finally, the distribution also includes a folder containing the source file of the converter from m single chain files to one parallel chains file, together with a toy-example to show how it works (see § 2.2.1).

The folder `CLTC_C_FILES_Method2` holds the core `C` files of `CLTC` appropriate if Method 2 is used. Most of these files should not been edited. Two "MCMC-specific" files that have to be edited by the user to define its MCMC sampler are in the same folder. The editable files given with the distribution provide an example implementing a Gibbs sampler for a multinomial benchmark example described in [3].

Folder CLTC_C_FILES_Method2:

GCLT_7.5.c	the main file for the current version
GlobalDecl.c	global declarations for the whole CLTC system
InOut.c	handling reading from and writing to files
MticaIntf.c	handling output interface to *Mathematica*
Simulation.c	contains the core functions of the program
McmcDecl.c	MCMC-specific declarations (editable)
McmcAlgo.c	MCMC sampler definitions (editable)

The folder CLTC_generic_Method1 holds a similar set of files for the generic version of CLTC, appropriate if Method 1 is used (i.e. convergence diagnostic based on simulated values from the MCMC sampler stored in a file) . **These files should not be edited**. An already compiled version, CLTC_1D_generic, is included on the Macintosh distribution for the current version.

Input and output sample files:

infile_sample	simulation tuning parameters (example)
outfile_sample	GCLT_7.5 program output for this example

Mathematica Notebook files:

CLTC_public_7.5.nb	Notebook for *Mathematica* 4
Sample_output.nb	Notebook giving a sample output

Executable files (Macintosh distribution only):

CLTC_1D_generic_7.5	generic version for Method 1 (version 7.5)
CLTC_7.5	sample version for Method 2
single2parallel	converter from single to parallel chains

PostScript files:

CLTCtechreport.ps	the companion preprint [2]
CLTCguide.ps	this user's guide

Folder single2parallel_Converter:

single2parallel.c	source file of the single to parallel chain converter
single2parallel	executable file (Macintosh only)
x.1	sample file, iterations for chain number 1
x.2	sample file, iterations for chain number 2
x.3	sample file, iterations for chain number 3
x.parallel	parallel chains file generated by the converter

2.2 Preparing for Method 1: control after the simulation

If you use Method 1 and you control one (or several) one-dimensional function of the Markov chain, then you don't have to edit any of the CLTC source files.

Under the Macintosh distribution you don't even have to compile anything. Let us assume that you want to control a set of K functions of the original chain, $\psi_k(\theta)$ for $k = 1, 2, \ldots, K$. In most situations K is the dimension of the parameter space and ψ_k is the k-th coordinate, i.e.

$$\theta_t = (\theta_{t,1}, \ldots, \theta_{t,K}) \in \mathbb{R}^K, \quad \text{and} \quad \psi_k(\theta_t) = \theta_{t,k} \in \mathbb{R}.$$

For a single chain implementation which is ran m times (the simplest method), the necessary steps are:

1. Implement your MCMC algorithm in the language of your choice (if your MCMC is not CPU-demanding, you can even use high-level languages such as `MATLAB`, `SciLab`,...)

2. Implement instructions within your MCMC simulation program to write the one-dim functions $\psi_k(\theta)$'s to plain text output files. A typical set of `C` instructions for that and for the k-th function is

```
FILE *fopen();    /* declaration */
char fname_k[30]; /* declaration for the k-th output file name */
FILE *Outfile_k;  /* declaration of pointer to k-th output file */

Outfile_k = fopen(fname_k,"w"); /* opening file in writing mode */

for (t=1; t <= n; t++)  /* loop over time */
   {
   your instructions for simulation here...
   fprintf(Outfile_k,"%f\n",psi_k(theta)); /*output current value*/
   }
```

   where `psi_k(theta)` is your implementation of your function $\psi_k(\theta)$, and `theta` is the current value of the multivariate parameter, θ_t.

3. **Run your MCMC sampler m times**: You have to control K functions $\psi_k(\cdot)$, $k = 1, \ldots, K$ of the original chain, but all the files should be generated simultaneously i.e. during these m runs (to save computing time). This means that each single run ℓ simulates the n iterations of the ℓ-th chain, and writes K output files, the k-th file containing the n iterations of the function $\psi_k(\theta_t^{(\ell)})$, $t = 1, \ldots, n$. Following this procedure, you should end up with $m \times K$ files.
   Initial distribution: It is important to start these m chains from different, uniformly distributed initial positions $\theta_0^{(\ell)}$. The good thing is to directly simulate the initial value within your MCMC program from a distribution "as dispersed and uniform as possible". See the example provided in the distribution for Method 2, which generates the initial values from the uniform distribution over the support defined by the user. Make sure also that your random generator is not giving you the same initial position for the m chains, as it can be the case if you use m times the same *seed* for the pseudo-random generator.

4. Generate K **"parallel chains" files**, where the k-th file contains the $m \times n$ values of $\psi_k(\theta)$ over the m i.i.d. chains. The format of this file

must be:

$$\begin{array}{|cccc|}\hline \psi_k(\theta_1^{(1)}) & \psi_k(\theta_1^{(2)}) & \cdots & \psi_k(\theta_1^{(m)}) \\ \psi_k(\theta_2^{(1)}) & \psi_k(\theta_2^{(2)}) & \cdots & \psi_k(\theta_2^{(m)}) \\ \vdots & \vdots & \vdots & \vdots \\ \psi_k(\theta_n^{(1)}) & \psi_k(\theta_n^{(2)}) & \cdots & \psi_k(\theta_n^{(m)}) \\ \hline \end{array}$$

i.e, the t-th row must contains the m observations at time t of the function ψ_k of the m i.i.d. chains, for $t = 1, \ldots, n$. The numerical values are expected by `CLTC` to be in standard floating point notation, as given, e.g., in `C` by the `fprintf` instruction similar to the one given above, using the format descriptor `"%f "` (giving 6 significant digits by default). Values in each lines are separated by a simple space character. The file `single2parallel.c` provides a `C` program for generating a "parallel chain" file from m single chain files (see below).

2.2.1 Using the "single to parallel" converter

The folder `Single2Parallel_Converter` contains the source file `single2parallel.c` (and a Macintosh application `single2parallel` in the Macintosh distribution) to perform step 4 above. This program will build a "parallel chains file" from m single chain files for you if your files follow the standard format given above (in step 2). Each of your single chain files must contain the n iterations of the one-dimensional function of the markov chain, one observation per line, in standard floating point notation (e.g., using `"%f\n"` in `C`). The folder holds a set of $m = 3$ single chain files of $n = 10$ iterations each, and the corresponding "parallel chains file", to provide an example of how it works. To use it, just compile the source file, launch the application, and answer the questions ("root name" for the input files, output files etc. . .).

Note that `single2parallel` works in a very simple way: it first successively opens the m single chain files, and load the observations into RAM. Then it writes out these $m \times n$ observations "in parallel" in the output file. This means that a large amount of RAM is necessary to process long sequences. For example, the compiled version given in the distribution can process $m = 50$ chains for up to $n = 20,000$ iterations. Since each value is coded as double precision (8 bytes), it requires about 8 Mo of RAM. If you need more than $20,000$ iterations, simply increase the size of `MAXIT` in the source file and recompile it.

2.3 Preparing for Method 2: simulation and control in the same program

For Method 2 described in § 1.4.2, you need to write the procedures needed to run your MCMC sampler in a text source file that will be compiled with the `CLTC` source files. This is done through the steps below.

2.3.1 Including the MCMC algorithm

1. Make a copy of the sample files `McmcDecl.c` and `McmcAlgo.c` from the folder `CLTC_C_FILES_Method2` since you will have to edit the original ones.

2. Edit the file `McmcDecl.c`: this file contains the MCMC-specific declarations, typically giving the parameter dimension for array declaration, and some cosmetic information like definition of names for parameter coordinates (which is what) and a name for your MCMC algorithm. These are not mandatory, they are just used for output. Below is an example of the editable lines in this file:

```
#define MAX_P_DIM 3   /* dimension for arrays declaration for */
                      /* the parameter, = param_dim + 1       */
int param_dim = 2;    /* parameter dimension, model-specific  */

/* enter below MCMC title and names for your coordinates */
char MCMCdef[] = "Gibbs sampler for the Multinomial benchmark.";
char *Coordname[] = {"","mu", "eta"};
```

   Here the MCMC sampler is two-dimensional, with $\theta = (\mu, \eta)$, so that the dimension of arrays must be 3 (due to the way arrays are defined in `C`, with the first item at coordinate `[0]`, which is not used in `CLTC`, so that θ_j is stored in `theta[j]`). For the same reason, the names of the coordinates begin with an empty string `""`. The other declarations in the file **should not be edited**.

3. Open `McmcAlgo.c` in a text editor and write in `ANSI C` the implementation of **one step** of your MCMC algorithm. The main procedure must be called `MCMC(current_theta,next_theta)`, since it is called with this name in the `CLTC` source files. The arguments are the current and next values of the multivariate parameter θ. Their names inside the procedure can be anything (e.g., `theta` and `update` as below), but their declaration must be like this:

```
void MCMC(theta,update)
double theta[MAX_P_DIM];      /* current parameter value */
double update[MAX_P_DIM];     /* updated value  */
{
your instructions here...
}
```

   In other words, if θ_t is the current position of a Markov chain associated to the parameter, then the "algorithm"

   `theta` $\leftarrow \theta_t$

   `MCMC(theta,update);`

   should return `update` = θ_{t+1}. Of course, this procedure can call other functions and procedures as needed; just put all these in the same text file `McmcAlgo.c` as for the example supplied. This step is the only one requiring some programming skill from the user. However, it should be straightforward, since the user is supposed to already have a program

source for the sampler he whishes to control. The file must contains its own declarations (random generator, parameters, procedure for reading observed data from a text file if the model needs data (e.g., in Bayesien setup, etc...). See the example provided as a template file in the distribution[1].

4. Edit the procedure `DrawInitTheta(theta)` in the same file `McmcAlgo.c`. Its purpose is to generate the initial positions randomly over the support defined by the user. See the example provided in the file for the 2-dim case.

5. **In some rare situations**, you may have to edit the file `Simulation.c` (in this case make a copy first). A call to your sampler,
`MCMC(current_theta,next_theta);`
is already included in the proper place in this file, and you usually don't even have to open this file. However, some particular MCMC algorithms may require some more specific instructions inside the main simulation loop to handle special situations. If it is the case, these instructions should take place in the main loop. You can locate the proper place in the file by searching for the comment

```
*BEGIN OF USER MODIFICATION SECTION*
```

to find where. See more instructions in the file.

CAUTION: do not make any change above or below the section in the file dedicated for user modification!!!

2.4 Compiling

After editing appropriate files, and whatever the method you choose, you have to compile the whole thing with a standard `ANSI C` compiler. You just have to compile the main program file `GCLT_7.5.c` that will include all the other files itself:

- On a `UNIX` workstation you may use, e.g., the gnu compiler `gcc` with the `-lm` option for math library. The typical command to generate an executable file called `CLTC` is:

```
gcc -lm -o CLTC GCLT_7.5.c
```

- On a Mac, existing compilers are Absoft C/C++, CodeWarrior distribution, MPW (Macintosh Programmer's Workshop)..., both for 68K or PowerPC chips under `MacOS 9` or older versions. Under `MacOS X`, a `C` compiler is included since it is a `UNIX`-based operating system (I did not check native versions of `CLTC` under `MacOS X` yet, but current version works under `Classic`). Here, you need to compile `GCLT_7.5.c` only if you are using Method 2.

- On a Wintel box, get a `C` compiler and do as for `UNIX`.

[1] The very good random generator `KISS` from Marsaglia and Zaman [8] is included and used in the sample file `McmcAlgo.c`. It is strongly recommended to use it for the simulations.

2.5 Input files and CLTC tuning parameters

Due to the need for the code to be compatible across a number of platforms, the input/parameters to the program are not specified through a graphical user interface, but via a simple text-mode which consists in:

1. defining the `CLTC` tuning parameters in the "**input file**", a text file with a specified format given below;
2. answering sequential questions at the beginning of the program, to tell `CLTC` the name of the input file, the data file (if any), the "parallel chains file" (if Method 1 is used), and the output file (for *Mathematica* post-processing).

Here is a sample input file for setting the control tuning parameters of `CLTC`:

```
mu0         2
StopRule    1
dtz_chain   1
ccoord      1
nmax        10000
nstep       100
fstcs       50
nmc         50
ncs         17
prob_tresh  0.0001
Support     0.8  3.5
```

The meaning of the parameters in the input file are:

`mu0`	initial distribution, 1=dirac, 2=uniform over the support (2 should always be used, 1 was merely for testing)
`StopRule`	stopping rule for convergence, 1 = $\mathcal{T}_M$, 2 = simultaneous (see [2]) (1 is less conservative and preferable)
`dtz_chain`	1 = request computation and output of the matrices for the discretized chain; 0 = not requested (see [2] and comment below)
`ccoord`	coordinate (marginal of θ) to control (needed for Method 2) should always be 1 for Method 1
`nmax`	total number n of iterations to run (Method 2) or read from the parallel chains file (Method 1)
`nstep`	number of iterations between each control stage
`fstcs`	iteration number of an additional first control stage ($<$ `nstep`)
`nmc`	number m of parallel markov chains
`ncs`	number of sets p in the partition of the support (see [2])
`prob_tresh`	probability treshold parameter ε (see [2])
`Support`	controlled region $\mathcal{A}$ for the marginal (see [2])

Tuning parameters important for the methodology: To fully understand the meaning of the tuning parameters $(\mathcal{A}, p, \varepsilon)$ that are crucial for the methodology, it if strongly recommended to read [2], § 4.1, where these are described in details. The mathematical notations that appear here are also defined in the reference papers.

The parameter `dtz_chain` also deserves more explanation: For a discrete, finite chain, it is possible to compute algebraically an estimate of the limiting variance of an indicator function of a singleton $\mathbb{I}_i$, where i is a point in the state space. This computation requires inversion of the estimated transition matrix of the chain, and is done in the *Mathematica* part of `CLTC` using matrices written in the `CLTC` output file. This computation helps to check stabilization of the estimate of the variance, by comparing it to the estimates of the variance after n steps, always provided by `CLTC`. Hence for a finite chain, it is a good idea to set `dtz_chain = 1`. For a continuous (general) chain, however, `CLTC` provides an approximation of the chain via a discrete "pseudo-chain" defined over the sets A_r, $r = 1, \ldots, p$, and an approximation of the limiting variance by inverting the approximated transition matrices against control times (see [2]). It should be pointed out that this approximation is not theoretically valid, and in addition, if $p =$ `ncs` is large, it requires in `CLTC` a large amount of extra material that has to be written in the output file, and processed by *Mathematica*. Hence for large p (e.g. $p > 100$), it is preferable to set `dtz_chain = 0`.

Note also that in this implementation, you specify with `nmax` the total number of iterations available in the parallel chains file (Method 1), or you select a maximum value `nmax` for the number of iterations to simulate (Method 2). If convergence is not achieved after `nmax` using Method 2, than you have to give a larger value for `nmax` and re-run the whole thing, which can be tedious if your MCMC is computationaly intensive. In this case, it is better to use Method 1, as suggested in § 1.4.1, point (5), since it is always possible to append more iterations to your single chain files, and to re-run `CLTC`, up to convergence.

2.5.1 The discrete case

In the discrete case, i.e. when each marginal $\theta_k \in \mathbb{N}$, it is preferable to define the support $\mathcal{A}$ and the sharpness p so that each set corresponds precisely to a single, or a set of integers. For example, if $\theta \in \{0, 1, \ldots, 10\}$, a good setting can be $\mathcal{A} = [-0.5, 10.5]$ with $p = 11$, so that each set A_r is of the form $[r-1/2; r+1/2]$, $r = 0, \ldots, 10$.

2.6 Launch CLTC

When `CLTC` comes up, it displays some information, and then asks few self-explicit questions. Below is an example using the `CLTC_1D_generic` version (Method 1):

```
MCMC CLT CONTROL FOR GENERAL MARKOV CHAINS
C pgm Version 7.5 - © D. Chauveau, 12/2001.
Generic: 1-dim MCMC sampler (or recoding) with obs from file

Enter file name for input parameters : in.a.17
Enter file name for Mathematica output : a.out
Enter output type (1=long, 0=short) : 0
Parallel chains file name : alpha
```

Note that the output type is a trigger between "short output type", which just displays summary information and is usually preferable, and "long output type" which displays much more computations in details, and is merely for debugging.

2.7 Program output

2.7.1 The C program output log

The log is what appears in the main `CLTC` window. You can redirect it to a file, or save it manually, but it is not saved automatically by `CLTC` in a file. After the questions above, `CLTC` prints the tuning parameters that have been read in from the input file:

```
MCMC CLT CONTROL FOR GENERAL MARKOV CHAINS
C pgm Version 7.5 - © D. Chauveau, 12/2001.
1-dim MCMC sampler (or recoding) with obs from file
(output file suitable for Mathematica processing).

SIMULATION PARAMETERS :
initial distribution method:             mu0 =      2
actual maximum nb of iterations:     nmaxint =  10000
nb of iterations between each control: nstep =    100
nb of control steps:                   nbpts =    101
nb of iterations before first control: fstcs =     50
nb of parallels MC to be run:            nmc =     50
nb of sets in partition:                 ncs =     17
control probability treshold:     prob_tresh = 0.000100
Support definition                          = (0.800000 , 3.500000)
Set size                                    = 0.158824
Stopping rule for convergence               = Max{CLT times}
ccoord=1
Controlled coordinate is x
computations related to discretized chains   = requested
```

Then, the parallel simulations are performed (Method 2), or read in from the "parallel chains file" (Method 1) together with the convergence diagnostic calculations, and results are displayed in the simulation log in a synthetic way. The "short output type" (selected at the beginning of the program) is preferable

for ordinary users. The short output gives information about the status of the sets A_r, i.e. of the partition of the support with respect to the approximate normality, against the control stages. The control stages have been defined by the user in the intput file, by the parameters `fstcs` and `nstep`, so that the control stages are performed at iterations

$$\texttt{fstcs}, \quad \texttt{nstep}, \quad 2 \times \texttt{nstep}, \quad 3 \times \texttt{nstep}, \ldots$$

up to the maximum number of iterations `nmax` selected. The status of the A_r's are given by a string of length p, of which the r-th letter gives the status of the corresponding A_r, which can be :

- `?` normality not reached yet for the statistic $S_n(\mathbb{I}_{A_r})/\sqrt{n}$ at control stage n
- `+` normality already accepted for the statistic $S_n(\mathbb{I}_{A_r})/\sqrt{n}$
- `-` normality not reached, but set discarded because of a too small observed probability ($< \varepsilon(t)$ where $\varepsilon(t) \to$ `prob_tresh`, see [2]).

Thus, a `?` turns to a `+` at the first control stage where normality has been accepted for the corresponding sum related to the corresponding A_r.

Below is an example of the output log of `CLTC`, which illustrates the fact that more and more sets A_r's are accepted for approximate normality when the number of iterations increase. Almost all the mass of the target distribution is actually "controlled" (in the sense of `CLTC`) at time $n = 1100$, except for the tails of the distribution. However, the sampler must be ran up to time $n = 6400$ in this example to accept normality in all the sets.

```
READING PARALLEL SIMULATIONS FROM FILE...
Output subsets status ([?]controlled, [+]accepted, [-]discarded):
n=     0: ????????????????
n=    50: ????????????????
n=   100: ????????????????
n=   200: ???+++?++????????
Normality reached for parameter & user function.
n=   300: ???++++++????????
n=   400: ???+++++++?+??????
n=   500: ???+++++++?+??????
n=   600: ???+++++++?+??????
n=   700: ???+++++++?+??????
n=   800: ???+++++++?++?????
n=   900: ?++++++++++++????
n=  1000: ?++++++++++++????
n=  1100: ?+++++++++++++???
etc...
n=  6300: ++++++++++++++++?
n=  6400: ++++++++++++++++?
--- SUCCESS IN STATIONARITY CHECKING ---
●●● Approximate normality reached at n = 6400 ●●●
Convergence times for each sets:
2400,800,800,100,100,100,200,100,100,800,300,700,800,1000,1300,4300,6400,
```

```
First entrance time is 100
Last entrance time is  6400
----- END OF STATIONARITY CHECKING -----
n =   6500: Parameter SW =0.965404
n =   6600: Parameter SW =0.965283
etc...
n =   9900: Parameter SW =0.964749
n =  10000: Parameter SW =0.962076
PARALLEL SIMULATION TERMINATED.

Normality reached at n=200 for the parameter.
Normality reached at n=200 for the user function.
```

If convergence is reached before `nmax` (i.e. approximate normality reached at least once for all the non-discarded statistics if `StopRule=1` has been selected in the input file, or reached simultaneously if `StopRule=2`), then `CLTC` output the time of convergence ($\mathcal{T}_M = 6400$ in the example above), the individual times of convergence of each set, and then just output the test statistic used in the program (the Shapiro-Wilk statistic) up to `nmax`. `CLTC` also gives the convergence times of the statistics associated to the function $h(\theta) = \theta$, and the so-called "user function" ($h(\theta) = \theta^2$ by default).

At the end of the computations, and even if convergence has not been reached, `CLTC` returns some useful information (feedback) about the pertinence of the control relatively to the choice of the tuning parameters $(\mathcal{A}, p, \varepsilon)$:

```
Mean Nb of jumps out of Support  : 28.820000
Estimated mass out of Support    : 0.002882
          >    left part         : 0.000134
          >    right part        : 0.002748
Estimated mass inside Support    : 0.997118

Estimated mass of controlled sets: 0.997118
Estimated non-controlled mass    : 0.002882
```

It is important to see the discussion in [2] or [4] about these indicators of the confidence of the control method.

2.7.2 The `CLTC` output file

The output text file generated by `CLTC` (`a.out` in the example in § 2.6) is not intended to be read/print or used directly by the user. This file is a *Mathematica* script file, which contains results of the convergence diagnostic such as empirical distributions over time, Shapiro-Wilk test statistics over sets, transition matrices of the "discretized chain" (if `dtz_chain = 1`, see § 2.5 and [2]), and calls to *Mathematica* functions and macros.

Using this file is optional. It delivers additional information about the MCMC control, such as graphics and histograms of the posterior distribution

with confidence intervals computed at approximate normality, and monitoring of the asymptotic variance. To obtain these additional diagnostics and graphics, this file must be processed by *Mathematica*, in a way described in the next section.

2.7.3 The Mathematica output

The *Mathematica* part of `CLTC` uses the NoteBook interface of *Mathematica*, which can handle in a single document plain text explanations, kernel definitions and procedures, computations, and graphics. These different sort of output are stored in hierarchical *cells* sectionning the document (see the *Mathematica* user manual).

The Notebook `CLTC_public_7.5.nb` included in the `CLTC` distribution is divided in two cells at the top level of the hierarchy: (i) "How to use this NoteBook", which contains more detailed explanations, together with definitions of control variables that may be changed by the user; and (ii) "Definitions and procedures/macros", that contains procedures and functions (i.e., *Mathematica* programs). **The cells containing definitions and programs are locked, and must not be changed in any way**. Indeed, the user don't have to edit or change anything in this NoteBook. The purpose of this NoteBook is to "prepare" your *Mathematica* session, so that *Mathematica* can handle your `CLTC` output files (i.e. to define some procedures so that the *Mathematica* kernel knows how to process the output files generated by `CLTC`). This process is detailed in the two sections below.

Preparing the Mathematica session

1. Start *Mathematica*
2. Open `CLTC_public_7.5.nb`
3. Evaluate the initialization cells with the menu command
   `Kernel` → `Evaluation` → `Evaluate initialization`
4. Eventually, set the "control variables" (`CILevel`, `OutputStats`...) that correspond to some tuning parameters of the graphical output (see more explanation in the corresponding cells in the NoteBook, the default settings should be fine for almost all purposes). Alternatively, you can copy these definitions in your personal NoteBook (see below), and change the values and re-evaluate these cells in your NoteBook.
5. Close `CLTC_public_7.5.nb` (don't save any change)

These steps take few seconds and have to be done only once in a session (this means that if you quit *Mathematica*, you have to re-execute the steps above when starting a new session).

Executing `CLTC` output files

1. Open a new, empty NoteBook (we will call it `my_results.nb`) that will store the results of `CLTC` for your application.
2. Tell *Mathematica* the directory that contains the output file(s) generated by `CLTC` for your application. Typical instruction for MacOS is `SetDirectory["your hard disk:your MCMC folder:"];`
3. Execute the output file generated by `CLTC`: the command is simply `<< output_file`

Then *Mathematica* process your output file and delivers textual and graphics information in cells in your NoteBook `my_results.nb`. There is a sample NoteBook `Sample_output.nb` in the distribution, which contains an example of the output generated by *Mathematica* when running an output file produced by `CLTC`. You can repeat step 3 above for all the output files generated by `CLTC`, in the same *Mathematica* session, and save all the results in `my_results.nb`.

The CPU time needed by *Mathematica* to process your `CLTC` output files vary with the size of the output files, which depend on several parameters, such as `ncs`, `nmax`, and `dtz_chain`. In a today computer, and for reasonable sizes (about 1 Mb), it takes 1 to 5 mn to process the file. You may have to increase the memory available for *Mathematica* in the MacOS platform.

2.8 Interacting with the program

As described in [2] and in § 2.5, the correct settings for the tuning parameters (in particular for $(\mathcal{A}, p, \varepsilon)$) can be easily found by trial-and-error. This trial-and-error process can be easily done with `CLTC`, particularly if you are using Method 1, since you don't have to re-run your MCMC sampler (providing that you already store a sufficient number of iterations in the parallel chains file).

The principle is to run `CLTC` several times using different sets of tuning parameters, **over the same parallel chains file**, and to check the resulting convergence diagnostics and pertinence of the control, up to achievement of good indicators of the convergence time and estimated controlled mass (see § 2.7.1 and [2]). The `C` part of `CLTC` alone can be used for that, but you can also keep an open *Mathematica* session for viewing the graphics and results, and switch between the results of `CLTC` and the *Mathematica* output.

If you are using Method 2, you can use `CLTC` in the same way, but the time needed to execute the `CLTC C` program depends on the complexity of your MCMC sampler.

The interactive way to use the `CLTC` software can thus be summarized by the steps below:

1. Start up *Mathematica* and "prepare" the session using the NoteBook `CLTC_public_7.5.nb` and following the guidelines in § 2.7.3
2. Set up your tuning parameters in an input file (see § 2.5)
3. Run the `CLTC C` program and check the results in the log file
4. Process the output file generated by `CLTC` in your *Mathematica* NoteBook and check the graphics, etc
5. If necessary, swith back to step 2 above (keeping your running Mathematica process) and retry with different tuning parameters, and so on.

During this trial-and-error process, you may discover that the preselected number of iterations (`nmax`) is too small to achieve convergence with good indicators of the pertinence of the method. This means that the MCMC sampler is "mixing too slowly", and that you have to increase the number of iterations to achieve convergence. Then:

- When using Method 1, the best thing to do is to **append** some more iterations to the parallel chains file(s) already computed (see § 1.4.1), and to set up the new value of `nmax` accordingly in the `CLTC` input file. Then go to step 2.
- When using Method 2, just increase the value of `nmax` in the `CLTC` input file and go to step 2.

Bibliography

[1] Brooks, S.P., and Roberts, G. (1998), Assessing convergence of Markov Chain Monte Carlo algorithms. *Statistics and Computing*, **8**(4), 319–335.

[2] Chauveau, D. and Diebolt, J. (1998) *An automated stopping rule for MCMC convergence assessment*. Rapport de Recherche RR-3566, INRIA Rhône-Alpes.

[3] Chauveau, D., Diebolt, J. and Robert, C.P. (1998) Control by the Central Limit Theorem. In *Discretization and MCMC convergence assessment* (C.P. Robert Ed.), Lecture Notes in Statistics no 135, Springer-Verlag, New York, **Chap. 5**, 99–126.

[4] Chauveau, D. and Diebolt, J. (1999), An automated stopping rule for MCMC convergence assessment. *Computational Statistics*, **14**, 3, 419–442.

[5] Gelfand, A.E. and Smith, A.F.M. (1990), Sampling based approaches to calculating marginal densities. *Journal of the American Statistical Association* **85**, 398–409.

[6] Geman, S. and Geman, D. (1984), Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.* **6**, 721–741.

[7] Gilks, W.R., Richardson, S. and Spiegelhalter, D.J. (1996), *Markov Chain Monte Carlo in practice*. Chapman & Hall, London.

[8] Marsaglia, G. and Zaman, A. (1993), *The KISS generator*. Tech Report, Dept of Statistics, University of Florida.

[9] Meyn, S.P. and Tweedie, R.L. (1993), *Markov chains and stochastic stability*. Springer-Verlag, London.

[10] Robert, C.P. (1996), *Méthodes de Monte Carlo par chaînes de Markov*. Economica, Paris.